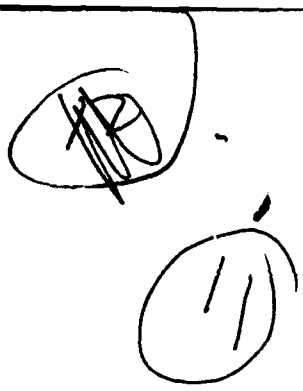

Can High Bandwidth and Latency Justify Large Cache Blocks in Scalable Multiprocessors?

Ricardo Bianchini and Thomas J. LeBlanc

Technical Report 486
January 1994

UNIVERSITY OF
ROCHESTER
COMPUTER SCIENCE

Can High Bandwidth and Latency Justify Large Cache Blocks in Scalable Multiprocessors?

Ricardo Bianchini and Thomas J. LeBlanc

ricardo@cs.rochester.edu, leblanc@cs.rochester.edu

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 486

January 1994

An important architectural design decision affecting the performance of coherent caches in shared-memory multiprocessors is the choice of block size. There are two primary factors that influence this choice: the reference behavior of application programs and the remote access bandwidth and latency of the machine. Several studies have shown that increasing the block size can lower the miss rate and reduce the number of invalidations. However, increasing the block size can also increase the miss rate by, for example, increasing false sharing or the number of cache evictions. Large cache blocks can also generate network contention. Given that we anticipate enormous increases in both network bandwidth and latency in large-scale, shared-memory multiprocessors, the question arises as to what effect these increases will have on the choice of block size.

We use analytical modeling and execution-driven simulation of parallel programs on a large-scale shared-memory machine to examine the relationship between cache block size and application performance as a function of remote access bandwidth and latency. We show that even under assumptions of high remote access bandwidth, the best application performance usually results from using cache blocks between 32 and 128 bytes in size. Using even larger blocks tends to increase the mean cost per reference, either because the miss rate increases or because the improvement in the miss rate is not enough to offset the increase in the miss penalty associated with larger blocks. We also show that modifying the program to remove the dominant source of misses may not help; the modified program could have a lower overall miss rate, but perform best with even smaller cache blocks. Since there are many factors that limit improvements in the miss rate with an increase in block size, and since the remote access bandwidth and latency limit the extent to which an improvement in the miss rate results in a lower mean cost per reference, we conclude that large cache blocks cannot be justified in most realistic scenarios.

This research was supported under ONR Contract No. N00014-92-J-1801 (in conjunction with the ARPA HPCC program, ARPA Order No. 8930) and NSF CISE Institutional Infrastructure Program Grant No. CDA-8822724. Ricardo Bianchini is supported by Brazilian CAPES and NUTES/UFRJ fellowships.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
Can High Bandwidth and Latency Justify Large Cache Blocks in Scalable Multiprocessors?

5. FUNDING NUMBERS
N00014-92-J-1801 /
ARPA HPCC Order 8930

6. AUTHOR(S)
Ricardo Bianchini and Thomas J. LeBlanc

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Dept.
734 Computer Studies Bldg.
University of Rochester
Rochester, NY 14627-0226

8. PERFORMING ORGANIZATION REPORT NUMBER
TR 486

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Information Systems
Arlington, VA 22217

ARPA
3701 N. Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution of this document is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

(see title page)

14. SUBJECT TERMS
cache block size; network bandwidth and latency; memory bandwidth and latency; scalable multiprocessors; locality enhancement

15. NUMBER OF PAGES
28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	UL

1 Introduction

The overhead of remote memory accesses is a major impediment to achieving good application performance on modern shared-memory multiprocessors. As processor speeds continue to improve at a dramatic rate, and as we anticipate building ever-larger machines, the relative importance of remote accesses will continue to grow. Shared-memory multiprocessors use hardware caches to keep data close to the processors that need it, and thereby reduce the average cost of a data access. The cache block size (that is, the size of coherency and fetching units) is an important design consideration affecting the performance of hardware caches. The choice of block size depends on the locality and sharing properties of applications, as well as the remote access latency and bandwidth. In this paper, we examine the relationship between these factors in the context of large-scale, network-based, cache-coherent, shared-memory multiprocessors.

Prompted by expected increases in interconnection network bandwidth (particularly with the use of optical networks), we consider whether or not increased bandwidth can be used to reduce the average cost of remote references through an increase in the cache block size. Larger cache blocks often result in reduced miss rates and, given sufficiently high bandwidth, can be transferred through the network at little additional cost. There are several factors that impose an upper bound on the cache block size however, and we would like to know what effect, if any, increases in bandwidth have on these limiting factors. We present an overview of these issues in section 2.

We use detailed execution-driven simulation of parallel programs on a large-scale shared-memory machine to examine the relationship between cache block size and application performance as a function of bandwidth. Our simulation methodology, performance metrics, and application workload are described in detail in section 3.

The experiments described in section 4 explore the effects of bandwidth on the choice of block size for each program in our application suite, using the miss rate and mean cost per reference as our main evaluation metrics. Our results show that block sizes between 32 and 128 bytes provide the best performance for our applications. Larger blocks usually result in an increase in the mean cost per reference, either because the miss rate increases or because the improvement in the miss rate is not enough to offset the incremental cost of fetching larger blocks.

In section 5 we consider whether improving the reference behavior of programs so as to reduce the miss rate produces a corresponding increase in the effective block size. We describe modifications to programs in our application suite that significantly improve the miss rate, and reconsider the effect of bandwidth on the choice of block size for these modified programs. The results of these experiments show that such modifications may not produce an effective increase in block size, and in any case increases in block size are limited by the ever-decreasing benefits of larger blocks.

In section 6 we introduce a simple analytical model of mean cost per reference, and use the model to generalize our experimental results relating bandwidth and block size, and to investigate the implications of increases in network latency.

We conclude, in section 7, that in nearly all cases, high remote access latency and bandwidth do not justify increasing the cache block size beyond 128 bytes, unless the network latency is extremely high.

2 Application and Architecture Issues Affecting Block Size

Factors that influence the choice of cache block size fall into two categories [Lee *et al.*, 1987]: (1) those that affect the miss rate of applications and (2) those that affect the cost of fetching a cache block.

The spatial and processor (sharing) locality of applications determines how miss rates vary as a function of the block size. Applications with good spatial locality usually benefit from using larger cache blocks, since most of the data in a cache block is likely to be referenced before it is evicted or invalidated. In the absence of write sharing of data, an increase in the block size reduces the miss rate until the *cache pollution* point [Eggers and Katz, 1989]. At that point, useless data begins to replace useful data in the cache, thereby increasing the miss rate.

The relationship between the cache block size and the miss rate has been studied extensively in the context of uniprocessors [Przybylski, 1990; Smith, 1987], but the miss rates of parallel programs do not always follow the same trends as sequential programs [Eggers and Katz, 1989]. Applications with good processor locality (i.e., coarse-grain sharing) typically favor large cache blocks since, for these applications, the true sharing miss rate goes down with an increase in block size. Applications with poor processor locality (i.e., fine-grain sharing) usually favor small cache blocks, so as to avoid false sharing [Eggers and Jeremiassen, 1991], and to avoid bringing data into the cache that will be invalidated before referenced.

In the best case (perfect spatial locality and coarse-grain sharing), doubling the size of cache blocks would cut the miss rate in half. Unfortunately, this best case scenario is extremely rare; increasing the block size typically causes more misses of one type while reducing the number of misses of another type. For example, if we classify misses as cold start, eviction, true sharing, false sharing, and exclusive request misses (caused by a write to read-shared data), then one can easily see that as we increase the block size, the number of cold start misses never increases, while the number of false sharing misses never decreases. The number of misses due to evictions, true sharing, and exclusive requests may decrease with an initial increase in block size, but will eventually reach a minimum, and then is likely to increase. As a result of these conflicting trends, an increase in block size may or may not improve the miss rate, depending on the reference behavior of the application, and the structure and size of the cache.

The choice of block size does not depend solely on miss rates of applications; we must also consider architectural parameters. In particular, remote access latency and bandwidth are important factors, as they determine the cost of fetching a cache block.[1] High remote access latency favors large cache blocks, since more data can be accessed with the same latency penalty. High remote access bandwidth also favors large cache blocks, since more data can be transferred for little extra cost. Large cache blocks can introduce network and memory contention problems however, since small packets generate less contention than large ones (assuming the same amount of data is transferred in both cases) [Agarwal, 1991].[2] Also, memory performance is affected by the block size; large blocks increase the memory busy time, thereby delaying contending processors.

[1]The latency of the memory is the time it takes to deliver the first word of data from the memory. The latency of the network is the time it takes to transfer a single word of data from source to destination. The bandwidth of the network (or memory) is the number of bytes transferred per cycle.

[2]In order to avoid this problem, large cache blocks could be transferred in several packets, and re-assembled at the destination. We do not exploit this technique in our simulations.

Increased network and memory bandwidth can reduce the cost of transferring large cache blocks, but do not change the dominant role of the miss rate. An increase in block size only improves performance when the larger blocks result in a lower miss rate. Even then, the decrease in the miss rate must be enough to offset the higher miss penalty associated with larger blocks.

Several researchers have studied the impact of cache block size on the miss rate and overall message traffic on small-scale, bus-based multiprocessors. Agarwal and Gupta [Agarwal and Gupta, 1988] found that 4-byte cache blocks generated the least bus traffic for their application programs. Eggers and Katz [Eggers and Katz, 1989] showed that, for applications with good per-processor locality, increasing the block size from 4 bytes up to 32 bytes improves the miss rate. They also showed that the effect on bus utilization depends on whether the improvement in the miss rate offsets the longer transfer time of larger blocks.

The results of these studies on small-scale, bus-based machines do not apply directly to scalable, network-based machines, which incorporate very different costs. Although a shared bus offers less communication bandwidth per processor than a direct-connect network, bus-based machines typically have a lower remote memory access latency than network-based machines. Limited bandwidth argues for small cache blocks to avoid contention, while a lower remote access latency reduces the penalty for extra transactions associated with smaller cache blocks. In addition, the broadcasting capability of a shared bus reduces the cost of invalidations, which means that any reductions in invalidation traffic achieved with larger cache blocks are not as significant in bus-based machines as in network-based machines.

In order to determine the feasibility of directory-based coherency schemes in network-based machines, Gupta and Weber [Gupta and Weber, 1992] studied the effect of the block size on the invalidation patterns of parallel programs. They found that data traffic goes up and coherence traffic comes down with an increase in block size, and that overall message traffic is minimized when the block size is 32 bytes. Since this study was mainly concerned with how changes in block size affect message traffic, it did not consider the corresponding effects on the miss rate or mean cost per reference, which have a more direct relationship to application running time. In addition, the argument in favor of 32-byte blocks is based on an assumption of limited bandwidth, since the negative effects of larger blocks are limited to an increase in the number of invalidations per write operation and increased message traffic.

Lee *et al* [Lee *et al.*, 1987] explored the performance effect of different cache block sizes as a function of network bandwidth, both in the presence and absence of explicit data prefetching. Their machine model assumes a multi-stage interconnection network, and a compiler-directed cache coherence scheme. They found that the optimal block size for multiprocessors is much smaller than for uniprocessors, and that explicit data prefetching encourages very small (4-byte) blocks. This study did not consider the dynamic sharing behavior of hardware cache coherence however, and therefore the performance of different cache block sizes on shared writable data could not be observed.

Dubnicki [Dubnicki, 1993] explored the effect of changes in cache block size on the mean cost per reference as a function of latency and bandwidth. He showed that the range of block sizes that minimizes the mean cost per reference of an application suite shifts upward (within the range of block sizes considered) with an increase in network bandwidth. For the particular application suite studied, the range shifted from 16..256 bytes at 20 MB/second to 64..256 bytes at 400 MB/second.

Dubnicki's work used trace-driven simulation, with traces collected on an 8-processor machine. We would expect such small-scale parallelism to result in less sharing and better locality than we would see on a large-scale machine, thereby favoring larger cache blocks. Also, his study assumed infinite caches and did not consider the effects of network contention, again favoring larger blocks. Since this study did not consider blocks larger than 256 bytes, the cumulative effect of these assumptions cannot be measured; no upper bound on block size is shown by the simulations.

None of these earlier studies definitively addresses the issue of block size on network-based shared-memory machines with high bandwidth. There are many complex factors (including the miss rate of applications, the cache size, and the latency and bandwidth of the machine), and previous studies either ignore one of more of these factors, or assumes a different architecture (such as bus-based machines) with very different costs. The question we address here is: what are the major impediments to effective increases in the block size on network-based high-bandwidth multiprocessors, can those impediments be alleviated, and how much does an increase in block size help?

3 Methodology and Workload

We are interested in exploring variations in bandwidth and cache block size in large-scale shared-memory multiprocessors, and therefore direct experimentation is not an available option. Thus, we use simulation for our studies.

3.1 Multiprocessor Simulation

We use an on-line, execution-driven simulator that exploits a mixture of interpretation and native execution to simulate unmodified MIPS R3000 object code. The simulator is divided into two parts, an event generator [Veenstra, 1993] and an event executor. The event generator simulates the processor and registers and calls the event executor on every memory reference. The event executor determines which processors block awaiting remote references and which processors continue to execute. We simulate events at the level of processor cycles; all simulation parameters and results are expressed in terms of processor cycles. Our event executor deals with all the major components of a parallel computing system: caches, the interconnection network, local memories, and directories.

We simulate a scalable direct-connected multiprocessor with 64 nodes. Each node in the simulated machine contains a single processor, cache memory, local memory, directory memory, and a network interface. Each processor has a 64 KB direct-mapped write-back cache. The cache block size is a parameter in our study. Caches are kept coherent using an implementation of the DASH protocol with release consistency [Lenoski *et al.*, 1990].

The simulator implements a full-map directory for controlling the state of each block of memory. Each node contains the directory for the memory associated with that node.

Throughout this paper we refer to the ensemble of addressable local memory and directory memory at each node as a "memory module." We simulate memory modules that queue requests (coming either from the cache or network interface) when the module is busy. Memory queues are assumed to be infinite. As should be the case for balanced architectures, we assume that the

Level	Path Width	Latency/Switch	Latency/Link	Bi-dir Link Bandwidth
Infinite	Infinite	2 cycles	1 cycle	Infinite
Very High	64 bits	2 cycles	1 cycle	1.6 GB/sec
High	32 bits	2 cycles	1 cycle	800 MB/sec
Medium	16 bits	2 cycles	1 cycle	400 MB/sec
Low	8 bits	2 cycles	1 cycle	200 MB/sec

Table 1: Network bandwidth levels used in simulated machine.

bandwidth of the memory module is equal to the unidirectional network link bandwidth (which is another parameter in our study). The latency of the memory module is 10 processor cycles.

The interconnection network is a bi-directional wormhole-routed mesh, with dimension-ordered routing. The network clock speed is the same as the processor clock speed. Switch nodes introduce a 2-cycle delay to the header of each message. The bandwidth of the network is a parameter in our study. In finite-bandwidth networks (derived from the Alewife cycle-by-cycle network simulator), contention for network links and buffers is fully captured. Each network interface has a queue for out-going messages, which is fed either by the cache or the memory module at the node. For comparison purposes we also implement an idealized, infinite bandwidth network, in which the path width is always larger than the size of messages.

Synchronization events do not generate memory or network traffic in our machine model, although they are used to maintain the relative timing of events. We ignore the traffic associated with synchronization so as to avoid having our results dominated by a poor implementation of locks or barriers.

3.2 Performance Metrics

For the most part our focus is on two different metrics: the miss rate and the mean cost per reference. The miss rate is computed solely with respect to shared references. That is, the miss rate is defined as the total number of misses on shared data divided by the total number of references to shared data. We classify misses using an extension of the algorithm in [Dubois *et al.*, 1993].

The mean cost per reference is defined as the number of each type of reference to shared data (hit or miss) times the average cost (a hit always takes 1 processor cycle to complete) divided by the total number of references to shared data. The mean cost per reference depends on the cost of remote accesses, which in turn depends on the latency and bandwidth of the machine. The levels of bandwidth we use are described in tables 1 and 2 (based on 100 MHz clocks). As stated earlier, the memory bandwidth is the same as the unidirectional network bandwidth.

The mean cost per reference metric has a direct relationship to running time. In a few instances we will relate running time to mean cost per reference as a way of exemplifying the effect of changes in the cost of each shared reference on overall performance. In those cases, running time accounts for **all** activities that occur during the simulated execution of a program.

Level	Latency	Cycles/Word	Memory Bandwidth
Infinite	10 cycles	0 cycles	Infinite
Very High	10 cycles	0.5 cycles	800 MB/sec
High	10 cycles	1 cycle	400 MB/sec
Medium	10 cycles	2 cycles	200 MB/sec
Low	10 cycles	4 cycles	100 MB/sec

Table 2: Memory bandwidth levels used in simulated machine.

Application	Shared Refs	Shared Reads (% of shared refs)	Shared Writes (% of shared refs)
`Mp3d`	21.1 M	60 %	40 %
`Barnes-Hut`	55.6 M	97 %	3 %
`Mp3d2`	39.3 M	74 %	26 %
`Blocked LU`	47.5 M	89 %	11 %
`Gauss`	64.5 M	66 %	34 %
`SOR`	20.7 M	85 %	15 %

Table 3: Memory reference characteristics on 64 processors.

3.3 Workload

Our application workload consists of six parallel programs: `Mp3d`, `Barnes-Hut`, `Mp3d2`, `Blocked LU`, `Gauss`, and `SOR`. `Mp3d` is a wind-tunnel airflow simulation of 30000 particles for 20 steps. `Barnes-Hut` is an N-body application that simulates the evolution of 4K bodies under the influence of gravitational forces for 10 time steps. `Mp3d` and `Barnes-Hut` are part of the SPLASH suite [Singh *et al.*, 1992]. `Mp3d2` is a version of `Mp3d` restructured for better cache behavior, as described in [Cheriton *et al.*, 1991]. `Mp3d2` and `Mp3d` use the same input. `Blocked LU` is an implementation of the blocked right-looking LU decomposition algorithm presented in [Dackland *et al.*, 1992] on a 384 × 384 matrix. `Gauss` is an unblocked implementation of Gaussian elimination that has been used in other studies, including [LeBlanc, 1988]. The input to `Gauss` is a 400 × 400 matrix. `SOR` performs the successive over-relaxation of the temperature of a metal sheet represented by two 384 × 384 matrices. Table 3 summarizes the distribution of shared references in our applications on a 64-processor machine.

As is the case with similar studies, simulation constraints prevent experimentation with "real life" input data sets. Simply reducing the input size to manageable levels without changing the cache size could produce unrealistic results however. Therefore the input data sizes used for our applications were chosen in tandem with our choice of cache size. We first determined input sizes that could be simulated in a reasonable amount of time, and then experimented with various cache sizes for those data sets. The cache size we ultimately selected, 64 KB, was chosen so as to avoid too heavy an emphasis on replacement misses; this cache size is the smallest that holds the working set of processors for our applications.

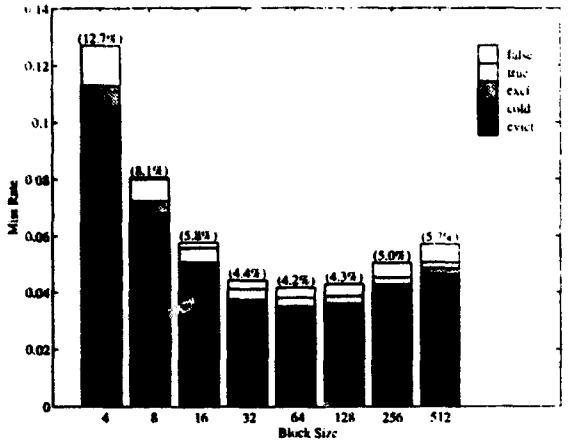

Figure 1: Miss rate of **Barnes-Hut**.

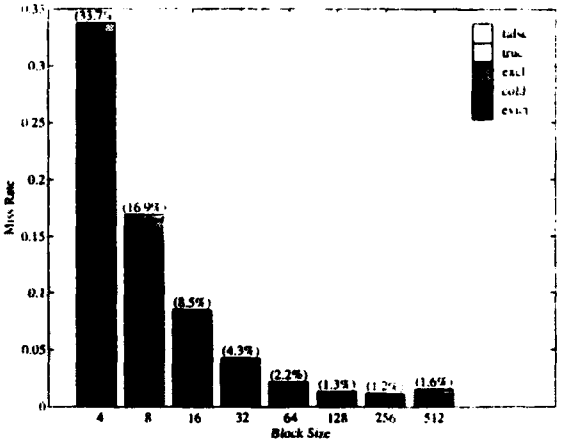

Figure 2: Miss rate of **Gauss**.

4 Minimizing Miss Rates and Mean Cost Per Reference

In this section we explore the effect of changes in block size on the miss rates and mean cost per reference (MCPR) of our application suite. We first use the miss rate to determine the optimal block size for an application under the assumption of infinite bandwidth. We then use the MCPR metric to determine the effect of remote access bandwidth and latency on the choice of block size.

4.1 Effect of Block Size on the Miss Rate

The block size that results in the minimum miss rate represents an upper bound on the size of cache blocks. Beyond this point, larger blocks simply increase the MCPR (and the running time of the application), regardless of the available bandwidth or the remote access latency. Given infinite bandwidth, the block size that minimizes the miss rate is optimal; smaller blocks incur larger penalties for transferring the same amount of data.

Figures 1-6 present the miss rates for each of our applications as a function of block size. The percentage at the top of each column represents the percent of all references to shared data that result in a miss; within a column misses are classified as either eviction, cold start, exclusive request, true sharing, or false sharing misses.

Figure 1 shows the miss behavior of **Barnes-Hut**. Even though the working set of a processor fits in its cache, the eviction miss rate is still a problem due to limited spatial locality and to the mapping of addresses in direct-mapped caches. The minimum miss rate occurs with 64-byte blocks; larger blocks increase the number of eviction and false sharing misses. The other categories of misses decrease with an increase in block size.

Figure 2 shows the miss behavior of **Gauss**. With 4-byte blocks the miss rate is very high (34%), but repeatedly doubling the block size (up through 128 bytes) continually cuts the miss rate in half. The minimum miss rate occurs when the block size is 256 bytes. These improvements in the miss rate are due to the excellent spatial and processor locality of the program. As with **Barnes-Hut**, the miss rate of **Gauss** is dominated by cache evictions. In particular, cache replacements are

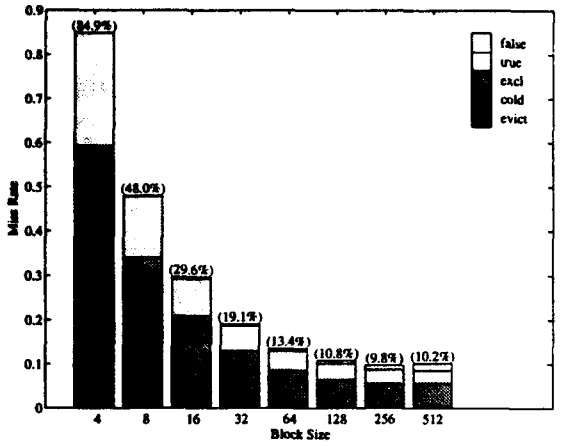

Figure 3: Miss rate of `Mp3d`.

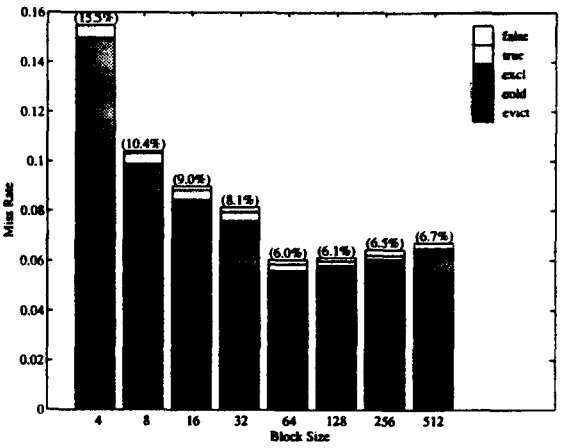

Figure 4: Miss rate of `Mp3d2`.

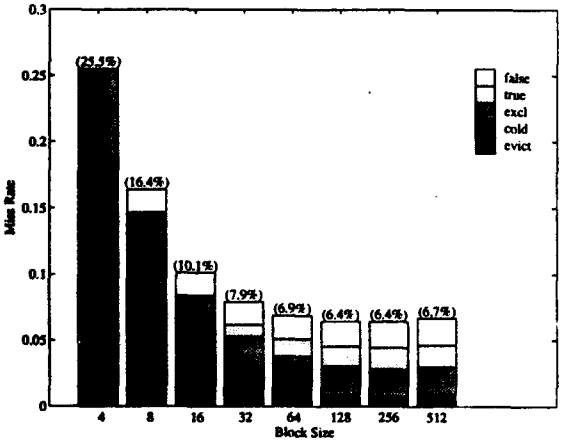

Figure 5: Miss rate of `Blocked LU`.

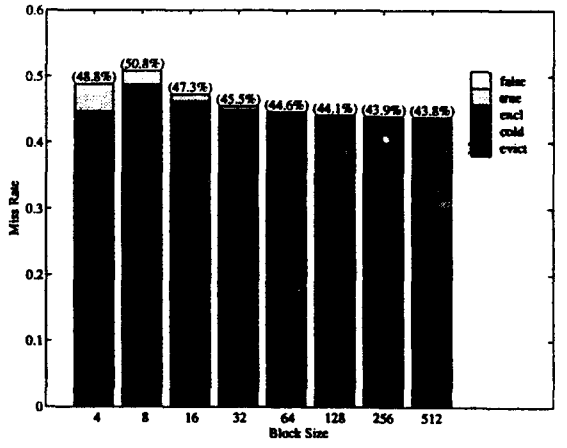

Figure 6: Miss rate of `SOR`.

responsible for the increase in the miss rate when moving from 256 to 512 byte blocks. The high eviction miss rate is due to poor temporal locality in accesses to the main matrix; each processor repeatedly references a large portion of the matrix for each row it is updating.

As seen in figure 3, Mp3d exhibits overall miss rate behavior similar to Gauss. For both programs increasing the block size between 4 and 256 bytes results in a decrease in the miss rate. The composition of the miss rate differs markedly between the two programs however. For Mp3d, false sharing is the limiting factor that precludes the use of 512-byte blocks. The miss rate is high regardless of block size, and in all cases is dominated by sharing-related misses.

Although Mp3d2 is an improvement of Mp3d, the two programs have very different memory referencing and miss rate behaviors. As expected, the miss rates for Mp3d2 are much lower than the corresponding miss rates for Mp3d. It is surprising however that the optimal block size for Mp3d is larger than the optimal block size for Mp3d2 (256 bytes instead of 64 bytes), even though Mp3d2 has much better locality of reference. In the case of Mp3d2, evictions dominate the miss rate, and the number of evictions increases with an increase in block size beyond 64 bytes. This example illustrates why even programs with good locality of reference may not be able to exploit large cache blocks.

Figure 5 presents the miss rate behavior of Blocked LU. As in Mp3d, the sharing-related misses dominate the miss rate. For the first time we can see significant amounts of false sharing, which is introduced with 8-byte cache blocks and remains fairly constant with larger cache blocks. Despite the false sharing, the minimum miss rate is achieved with reasonably large cache blocks (128 or 256 bytes).

SOR (figure 6) is interesting in that the miss rate is dominated by replacement misses, but those misses are insensitive to the block size. The minimum miss rate is achieved with the largest block size (512 bytes). The reason for this anomalous behavior is that SOR manipulates two matrices, where the memory size of each matrix is a multiple of the processor cache size. Since each processor modifies the same row indices in both matrices, rows from one matrix collide with the corresponding rows in the other matrix in the direct-mapped cache. In section 5 we describe the effects of modifications to SOR designed to eliminate this cache mapping problem.

In summary, for the majority of our applications, the minimum miss rate is achieved by using cache blocks between 64 and 256 bytes in size. There is no single type of miss that places this upper bound on cache block size: false sharing, true sharing, exclusive misses, and eviction misses are all significant contributors to the miss rate of some applications. Thus, we can conclude that for these application programs, enormous increases in bandwidth could not be used to reduce the average cost of remote references through a substantial increase in the block size.

4.2 Effect of Block Size on the Mean Cost Per Reference

In the absence of latency and bandwidth considerations, the miss rate of applications dictates the choice of block size. In practice however, the remote access latency and bandwidth also constrain the choice of block size. Thus, we cannot simply choose the block size that results in the lowest miss rate; we must consider whether any improvement in the miss rate that occurs with an increase in the block size offsets a corresponding increase in the miss penalty, which is dictated by the bandwidth and latency of the machine. We will examine this issue by considering how changes in the block size affect the MCPR.

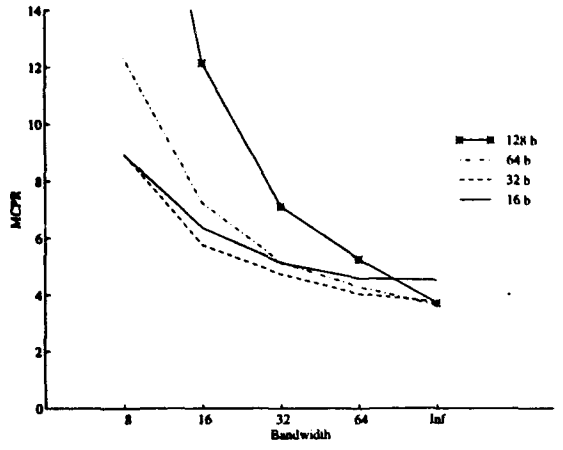

Figure 7: MCPR of Barnes-Hut.

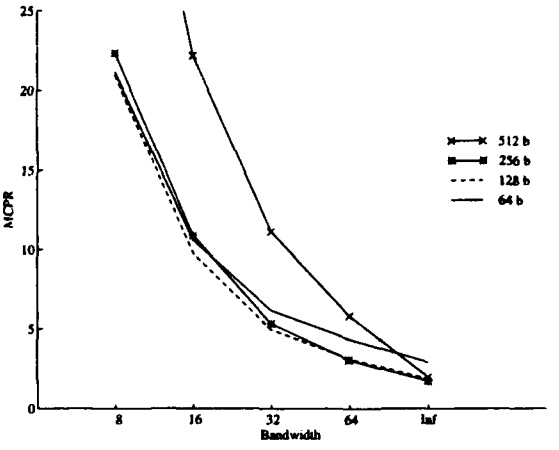

Figure 8: MCPR of Gauss.

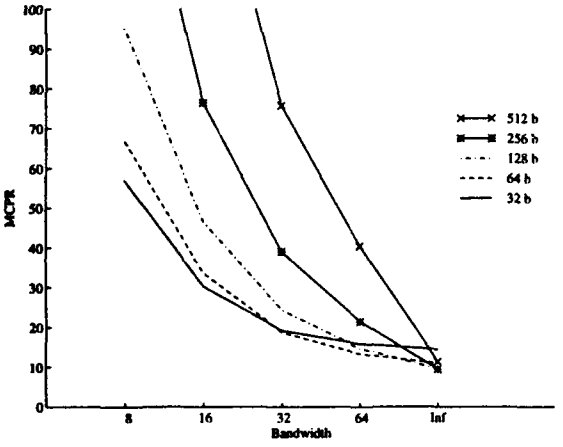

Figure 9: MCPR of Mp3d.

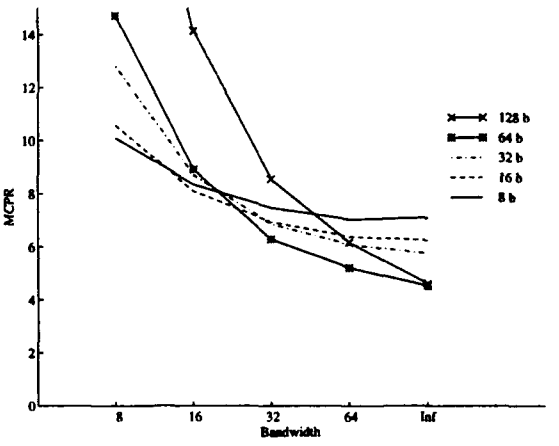

Figure 10: MCPR of Mp3d2.

Figures 7-12 present the mean cost per reference for our applications, as a function of the block size and the available (network and memory) bandwidth. For each application we only present data for the range of block sizes that results in the lowest MCPR.

Figure 7 presents the MCPR for Barnes-Hut. Across a wide range of bandwidth levels, 32-byte cache blocks result in the lowest MCPR. Larger blocks offer competitive performance only at very high levels of bandwidth, even though 64-byte blocks produce the minimum miss rate. At the lowest level of bandwidth, the performance of 16-byte blocks is comparable to the performance of 32-byte blocks. These results suggest that the improvement in the miss rate that occurs when increasing the block size from 16 to 32 bytes (5.8% down to 4.4%) is sufficient to offset the corresponding increase in the miss penalty even at very low bandwidth levels. On the other hand, the improvement in the miss rate that occurs when increasing the block size from 32 to 64 bytes (4.4% down to 4.2%) cannot offset the corresponding increase in the miss penalty, unless infinite bandwidth is available.

The MCPR of Gauss (figure 8) and Barnes-Hut exhibit roughly the same behavior. In both

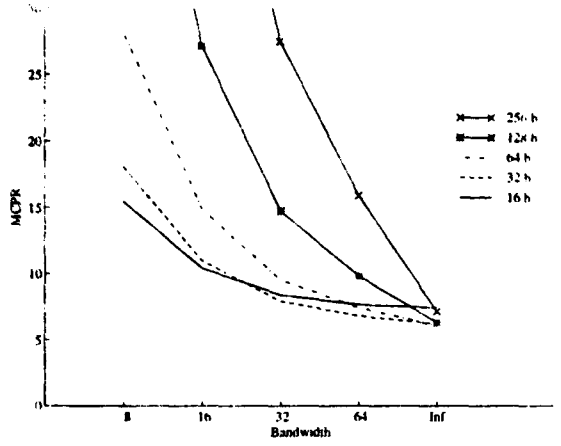

Figure 11: MCPR of **Blocked LU**.

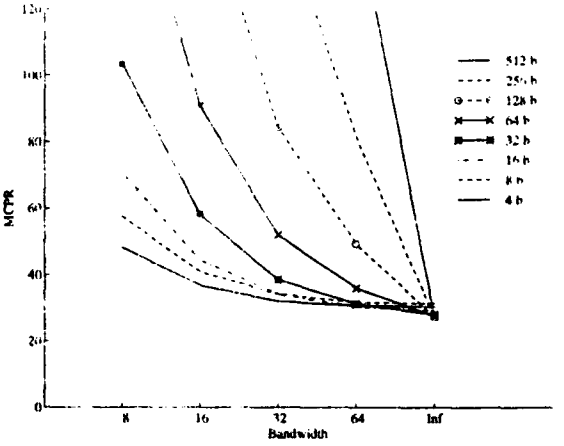

Figure 12: MCPR of **SOR**.

cases, a single block size (128 bytes for **Gauss**) offers the best performance over a wide range of bandwidth levels. Also, this block size is not the one that minimizes the miss rate (256 bytes for **Gauss**). The main difference between **Gauss** and **Barnes-Hut** in these figures is that bandwidth has a much greater impact on the MCPR of **Gauss**. Unlike **Barnes-Hut**, **Gauss** exhibits memory and network contention, so increasing the bandwidth of these system components drastically improves overall performance. Thus for **Gauss** using 256-byte cache blocks, an 8-fold increase in bandwidth improves the MCPR by a factor of 7, and the running time by a factor of 5.

As seen in figure 9, three different block sizes perform best for **Mp3d**, depending on the available bandwidth. At low and medium bandwidth levels, 32-byte blocks perform best, which is quite surprising since the miss rate with 32-byte blocks is almost twice the miss rate with 256-byte blocks (19.1% vs. 9.8%). With high bandwidth, 64-byte cache blocks produce the lowest MCPR. At infinite bandwidth, larger blocks (128 and 256 bytes) prevail.

Figure 10 shows a similar trend for **Mp3d2**: small cache blocks (8 bytes) perform best with low bandwidth, slightly larger blocks (16 bytes) perform best with slightly higher bandwidth, and even larger blocks (64 bytes) perform best in all other cases. Also, for the first time the block size that produces the minimum miss rate also produces the minimum MCPR for practical levels of bandwidth. The reason for this is that the optimal block size (64 bytes) improves the miss rate by 35% over the next smaller block size, which is enough to offset the higher miss penalty associated with the larger blocks. For the other applications the improvement in miss rate between the optimal block size and the next smaller block size is at most 10%, which in most cases is not enough to offset the higher miss penalty.

The best cache block size for **Blocked LU** also depends on the available bandwidth. For machines with low or medium bandwidth, a block size as small as 16 bytes minimizes the MCPR (and therefore the running time), as seen in figure 11. For higher levels of bandwidth, the best performance is achieved with a block size of 32 bytes, which is much smaller than the block sizes that minimize the miss rate (128 and 256 bytes). Note that although 128-byte blocks and 256-byte blocks result in the same miss rate, the MCPR of 256-byte blocks is always higher, even under infinite bandwidth. In this particular case, program execution with the larger cache blocks happens

to result in more queueing at the memory modules, which significantly increases the miss service time.

Under practical levels of bandwidth, given two block sizes that produce the same miss rate, we would expect the smaller block size to yield a lower MCPR, due to a lower miss penalty. Given infinite bandwidth we would expect the two block sizes to produce comparable MCPRs, except when a change in block size affects the interleaving of remote requests in such a way as to introduce or alleviate memory contention, with a corresponding impact on MCPR.

As seen in figure 12, SOR is clearly an exception to the general trend that higher bandwidth encourages the use of larger cache blocks. At any practical level of bandwidth, the block size that minimizes the MCPR in SOR is 4 bytes. The miss rate of SOR is relatively constant for every block size we considered, and the minor fluctuations in miss rate cannot compensate for the higher miss penalty associated with larger blocks.

Summarizing the results in this section, we showed that several factors contribute to the miss rate of our applications, any one of which can limit effective increases in the block size. In addition, we showed that bandwidth limitations further constrain the size of cache blocks. For our application suite, block sizes between 32 and 128 bytes provide the best overall performance even under the assumption of relatively high bandwidth. It is somewhat surprising that no amount of bandwidth suffices to justify blocks much larger than this. In the next section, we consider whether more carefully tuned application programs can exploit larger cache blocks, so as to more fully utilize expected improvements in bandwidth.

5 Increasing the Effective Block Size by Improving Locality

The results of the previous section suggest that many shared-memory applications cannot benefit from block sizes larger than 64 or 128 bytes. The question then becomes whether or not locality-enhancing techniques directed at reducing the impact and extent of the dominant class of cache misses in a program would allow for larger cache blocks to be used. In order to investigate this issue, we modified three programs in our application suite so as to alleviate the dominant source of misses in each program.

The first program we modified is SOR. Recall that this program suffers from interference in the mapping of addresses to cache locations; a processor must frequently replace data in the cache, even though the size of its working set is smaller than the size of its cache. To remove this source of eviction misses in SOR we added padding between the two matrices used in the program, so as to ensure that no two rows accessed by a single processor map to overlapping sets of cache blocks. We call the resulting program **Padded SOR**.

Figure 13 shows the miss rate for the modified program. As seen in the figure, our program modification completely eliminates evictions as a source of misses, and thereby dramatically improves the miss rate. As a side effect of reducing evictions, we also eliminated most of the exclusive request transactions required to regain ownership of evicted blocks. As a result, the number of exclusive request transactions is much lower and is now dependent on the block size. Together these effects lower the minimum miss rate from 43.8% to 0.1%, demonstrating the near perfect spatial locality and limited sharing exhibited by **Padded SOR**.

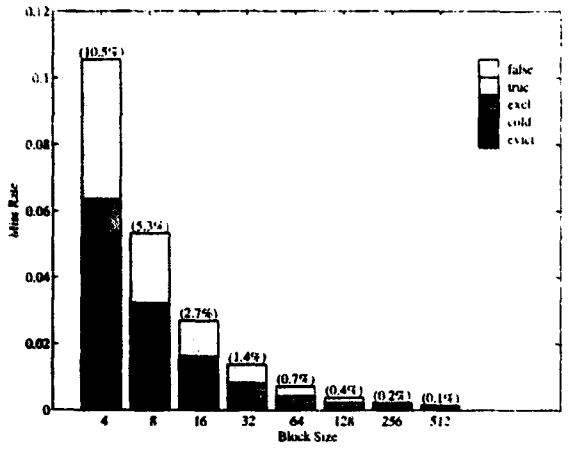

Figure 13: Miss rate of **Padded SOR**.

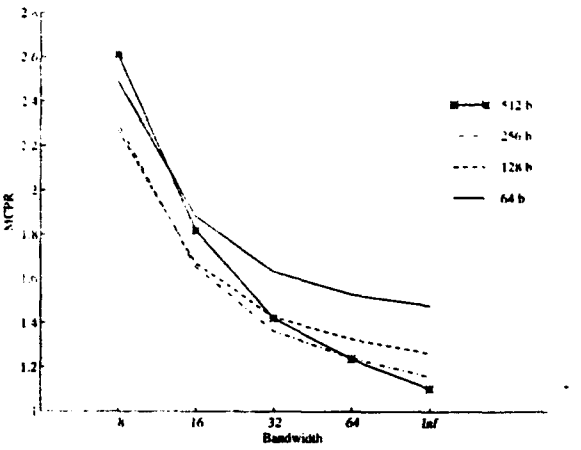

Figure 14: MCPR of **Padded SOR**.

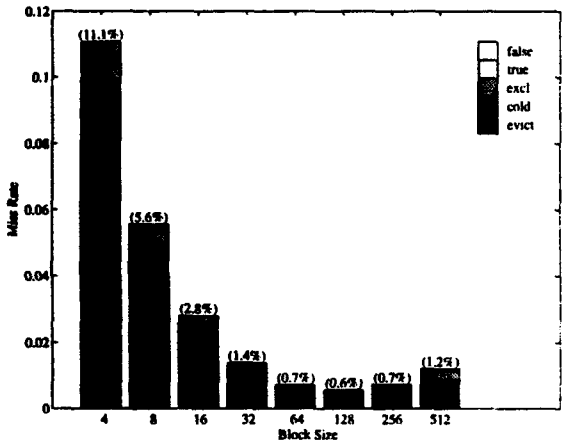

Figure 15: Miss rate of **TGauss**.

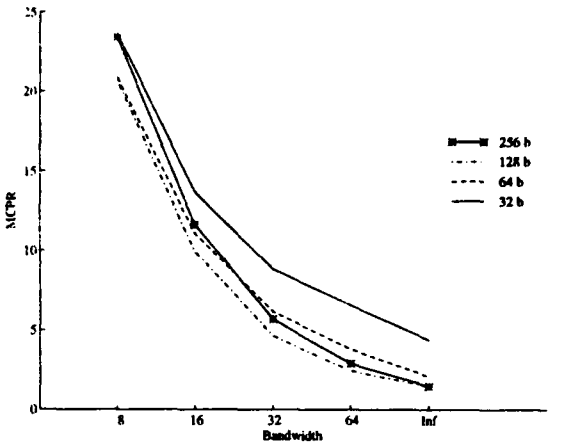

Figure 16: MCPR of **TGauss**.

For both **SOR** and **Padded SOR**, the minimum miss rate is achieved with 512-byte blocks. Nevertheless, as seen in figure 14, under most practical levels of bandwidth, 256-byte blocks produce the lowest MCPR for **Padded SOR**, while 4-byte blocks produce the lowest MCPR for **SOR**. For **Padded SOR**, the substantial improvements in the miss rate that result from increasing the block size up to 256 bytes offset the corresponding increase in the miss penalty, whereas the relatively minor improvements in the miss rate of **SOR** offered by blocks larger than 4 bytes do not offset any increase in the miss penalty.

Next, we modified **Gauss** to improve its temporal locality, and thereby reduce the number of eviction misses. We modified the program so that each processor reads a pivot row once, updates all of its local rows based on that pivot row, and then reads the next pivot row. The resulting program is called **TGauss**.

By comparing the miss rates of **Gauss** (figure 2) and **TGauss** (figure 15) we can see that this modification is very successful at reducing the number of replacement misses. In addition, the

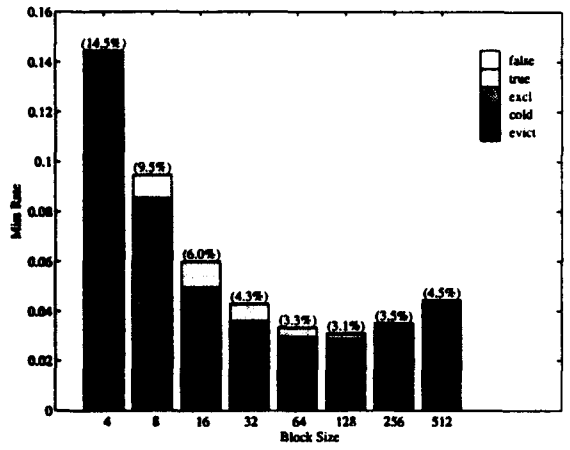

Figure 17: Miss rate of **Ind Blocked LU**.

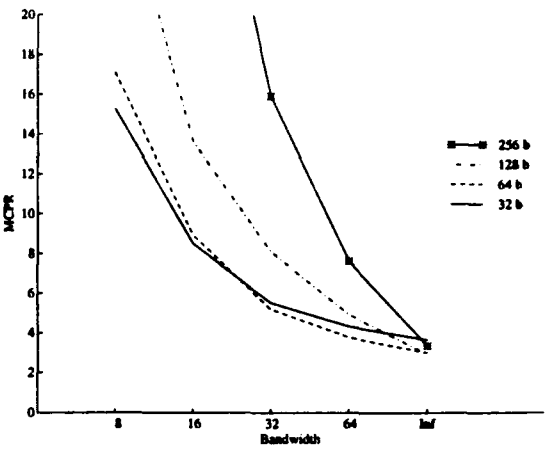

Figure 18: MCPR of **Ind Blocked LU**.

overall miss rate of **TGauss** is a factor of 3 smaller than the miss rate of **Gauss** for most block sizes. It is therefore surprising to see that the minimum miss rate for **TGauss** occurs with 128-byte blocks, whereas the minimum miss rate for **Gauss** occurs with 256-byte blocks. The composition of misses is different for the two programs, although evictions are the driving force in the overall miss rate in both cases.

Figure 16 shows that even though the upper limit on effective block size for **TGauss** is smaller than the upper limit for **Gauss** (128 vs. 256 bytes), both programs achieve their lowest MCPR with 128-byte cache blocks regardless of bandwidth. Thus, in this case, a program modification that improves locality does not increase the size of cache blocks that can be utilized effectively.

Our last program modification involves **Blocked LU**. Recall that the miss rate of this program is dominated by sharing-related misses for block sizes larger than 16 bytes. We modified **Blocked LU** to produce **Ind Blocked LU**, using *indirection* [Eggers and Jeremiassen, 1991] to reduce the number of true, false, and exclusive request misses. We added one level of indirection to each access to shared data, and stored the shared data in separate memory regions. Although references to shared data require two memory accesses instead of one (one to read the pointer to the data, and the other to read the data), writes to different shared data locations reference different memory regions and therefore don't conflict. In order for **Ind Blocked LU** to execute faster than **Blocked LU**, the lower miss rate must more than compensate for the additional references (one of which, the reference to the pointer, is usually a cache hit).

The miss behavior of **Ind Blocked LU** is shown in figure 17. As expected, the improvement in sharing-related misses is significant, although the number of cold start and eviction misses increases somewhat. The optimal block size is the same for both **Blocked LU** and **Ind Blocked LU** (128 bytes) however. Larger blocks increase the number of evictions, and thereby cause the miss rate of **Ind Blocked LU** to go up. Evictions play a larger role in the miss rate of **Ind Blocked LU** because the use of pointers for indirection effectively increases the working set size of processors.

The MCPR for this application is shown in figure 18. Once again, we see that the best block size depends on the available bandwidth. Given low bandwidth, 32-byte blocks outperform all others.

For all other levels of bandwidth, 64-byte blocks perform best. Thus, for high levels of bandwidth **Ind Blocked LU** favors a larger block size (64 bytes) than **Blocked LU** (32 bytes).

In summary, we modified three of our programs so as to improve the miss rate with an eye towards increasing the block size. While all of our program modifications were successful at reducing the miss rate, the resulting improvement in locality did not always affect the choice of block size. In two cases (**Padded SOR** and **Ind Blocked LU**) the optimal block size (that is, the block size that results in the minimum miss rate) remained the same, while in the other case (**TGauss**) it actually shrank. The block size that produced the lowest MCPR remained unchanged in one case (**TGauss**), grew slightly in another case (**Ind Blocked LU**), and grew enormously in the third case (**Padded SOR**).

These examples clearly show that improvements in locality of reference may not translate to effective increases in the block size. In fact, the miss rates of the modified programs are so small that there is little reason to believe that further improvements in locality could help justify larger cache blocks. In the next section, we use analytical modeling to argue that the upper bound on effective block size exhibited by our programs is not likely to be exceeded by other application suites, including those exhibiting good locality.

6 An Analytical Model of Mean Cost Per Reference

In this section, we present a simple model of MCPR in k-ary n-cube architectures based on Agarwal's model of network communication [Agarwal, 1991]. Our model relates the miss rate of the application and the remote access latency and bandwidth of the multiprocessor. We first present the model and validate it by comparing the model's predictions to the results in the previous sections. We then use the model to determine the improvements in miss rate required to justify an increase in block size, and to examine the effect of remote access latency on MCPR and the choice of block size.

6.1 The Model and Its Validation

We make the following simplifying assumptions in our model:

- Network links are bi-directional and there are no end-around connections.
- Network messages have randomly chosen destinations.
- The probability that a processor initiates a network transaction on any given cycle is uniform across processors.
- Remote requests are satisfied with two-party transactions. That is, a cache miss is satisfied by a single request/reply transaction between the requesting processor and the home node of the requested cache block; no other nodes are ever involved. This assumption is based on our experience with simulations of the DASH coherence protocol which show that two-party transactions dominate.

We define the mean cost per reference to be the number of cache hits times the average cost of a hit plus the number of cache misses times the average miss service time. Thus, the mean cost per reference (MCPR) for a block of size b is:

$$MCPR_b = h_b \times T_h^b + m_b \times T_m^b$$

where h_b is the hit rate with blocks of size b, T_h^b is the average time to service a cache hit (which we assume is 1 cycle), m_b is the miss rate with blocks of size b, and T_m^b is the average miss service time for blocks of size b. To simplify our notation, we will omit the dependence of MCPR, the hit rate, the miss rate, and the service times on the block size in those cases where the block size is fixed.

T_m depends on the time spent in the network, the time spent waiting in the memory queue, and the actual memory service time. More specifically,

$$T_m = 2\left(L_N + \frac{MS}{B_N}\right) + \left(L_M + \frac{DS}{B_M}\right)$$

where L_N and L_M are the average latency at the network and memory (including the average time waiting in the memory queue), B_N and B_M are the path widths (representing bandwidth) of the network and the memory, MS is the average message size, and DS is the average number of bytes provided per request by the memory modules.

The average network latency can be calculated in two ways, depending on whether or not we model contention. In the absence of contention,

$$L_N = D \times T_s + (D-1)T_l$$

where D is the average distance between source and destination, T_l is the message header delay per communication link, and T_s is the delay per switch node. With randomly chosen message destinations, $D = n \times k_d$ for k-ary n-cubes. With bi-directional links and no end-around connections, k_d, the average distance in a single dimension, is $(k - \frac{1}{k})/3$ [Agarwal, 1991].

In the presence of contention, the average network latency is

$$L_N \approx D\left[T_l + T_s + \frac{\rho \times \frac{MS}{B_N}}{(1-\rho)} \frac{(k_d - 1)}{k_d^2}\left(1 + \frac{1}{n}\right)\right]$$

where ρ, the average channel utilization, is $\mu \times \frac{MS}{B_N} \times k_d/2$; and μ, the probability of a network request on any given cycle from a processor, is $\frac{2}{(T_m + (1/m))}$. For further details on the network contention model, see [Agarwal, 1991].

To verify the accuracy of the model, we compare the model's predictions to the detailed simulation results presented in the previous sections. We instantiate the model using data derived from simulations that assume infinite bandwidth. These simulations do not require a detailed cycle-by-cycle simulation of the network, and therefore can be used to provide inputs to the model easily.

This approach assumes that the model parameters we collect from simulations with infinite bandwidth (such as the miss rate and the average communication distance) do not change significantly under variations in bandwidth; our experiences with the simulations described earlier suggest this is a valid assumption in most cases.

To instantiate the model, we collect the following statistics from simulations with infinite bandwidth: the miss rate, the average size of network messages, the average service time of the memories (including queue delays), the average number of bytes provided by the memories per operation, and the average distance traveled by network messages. The other architectural parameters used in the simulations are kept constant at the values used earlier: 64 processors, mesh topology, minimum service time of 10 cycles, and a delay per link and per switch node of 1 and 2 cycles, respectively.

We use the statistics from our simulations with infinite bandwidth to instantiate the model and predict the MCPR for a variety of block sizes as a function of bandwidth. We can then compare the MCPR predicted by the model (M) to the MCPR produced via detailed simulations (S). Figures 19-22 present this comparison for some of our application programs.

As seen in figure 19, the model accurately predicts the MCPR for **Barnes-Hut** over a range of block sizes and bandwidth levels. The MCPR predicted by the model is within 10% of the MCPR derived from the detailed simulations for all block sizes and bandwidths shown in the figure. The model is just as accurate in predicting MCPR for **mp3d2** (not shown), and is almost as accurate for **Padded SOR** (figure 20), except that the model predictions for **Padded SOR** with 16-byte cache blocks are 20-30% lower than the MCPRs produced via simulation. Given high bandwidth or small cache blocks the model is accurate for **SOR** (figure 21), **mp3d** (not shown), and **Blocked LU** (not shown), but is off by a factor of 2 or more when there is very low bandwidth and large cache blocks. Similarly, the model produces fairly accurate results for **Gauss** (figure 22), **TGauss** (not shown), and **Ind Blocked LU** (not shown) with large blocks and high bandwidth, but the predictions for small blocks and low bandwidth are too low by a factor of 2 or 3.

In those cases where the model and simulation results differ, the model fails to accurately account for network and memory contention. Contention may arise any time we have very low bandwidth, which explains why the errors in predictions for **SOR**, **mp3d**, and **Blocked LU** occur at low bandwidth levels. Contention may also arise when a non-uniform distribution of references results in a hot spot, as occurs in **Gauss**, **TGauss**, and **Ind Blocked LU**. Contention can be alleviated by high bandwidth, which explains why all of the predictions at high levels of bandwidth are fairly accurate. Contention may also be alleviated by large cache blocks, if the larger blocks cause a significant reduction in the miss rate (as in the case of **Padded SOR**). On the other hand, larger cache blocks can cause more contention (i.e., **SOR** and **Blocked LU**) since network contention increases dramatically with an increase in average message size.

Despite these limitations, the model is fairly accurate when predicting MCPR under high bandwidth, or when the program exhibits a uniform distribution of references. In the remainder of this section we will use the model to predict MCPR under high bandwidth levels.

6.2 Quantifying the Benefits of Large Cache Blocks

In the previous sections we claimed that cache blocks larger than 128 bytes are not likely to improve application performance. There are two main reasons for this: larger blocks can increase the miss rate due to sharing behavior or eviction misses, and larger blocks increase the miss penalty without

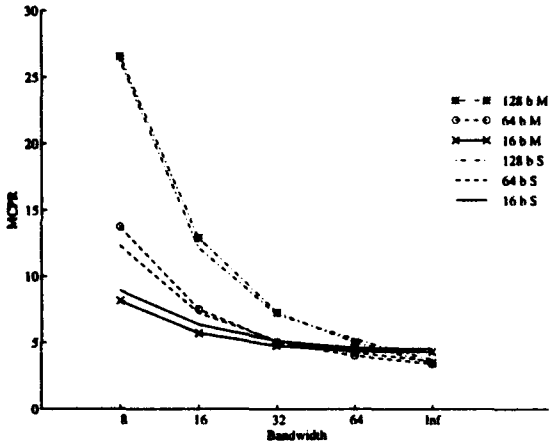

Figure 19: Simulated vs. predicted MCPR of **Barnes-Hut**.

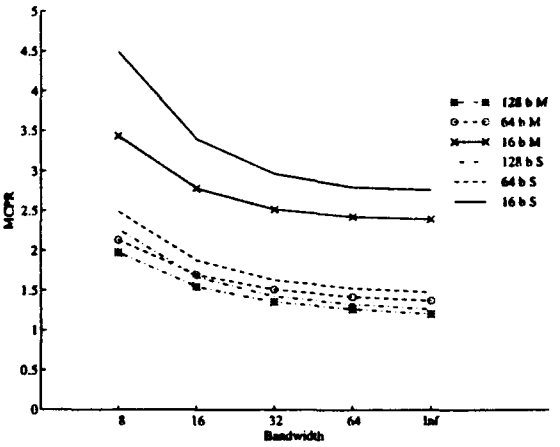

Figure 20: Simulated vs. predicted MCPR of **Padded SOR**.

necessarily reducing the number of misses significantly. We will now use our analytic model to show why very large cache blocks are unlikely to improve performance even when programs exhibit good locality and the architecture provides high remote access bandwidth.

We assume that block sizes are a power of two. The evaluation metric we use is MCPR. Thus, we should increase (double) the block size from b to $b \times 2$ only if the MCPR produced by using block size $b \times 2$ is less than the MCPR produced by using block size b. Assuming that the time to service a cache hit is one cycle, the larger block size is preferable when the following holds:

$$(1 - m_{2b}) + m_{2b} \times T_m^{2b} < (1 - m_b) + m_b \times T_m^b$$

where m_i is the miss rate when the block size is of size i, and T_m^i is the time to service a miss for a block of size i. If we assume

$$T_m^b = 2\left(L_N + \frac{MS}{B_N}\right) + \left(L_M + \frac{DS}{B_M}\right),$$

and if we also assume that b is large enough that the message headers are a small percentage of the bytes transferred, and that the proportion of exclusive request misses (in which no real data is transferred) with respect to the total number of misses is roughly maintained when doubling the block size, we can express T_m^{2b} as follows:

$$T_m^{2b} \approx 2\left(L_N + \frac{2 \times MS}{B_N}\right) + \left(L_M + \frac{2 \times DS}{B_M}\right)$$

Doubling the block size improves the MCPR if the improvement in miss rate offsets the increase in miss penalty, which means that the following holds:

$$m_{2b}\left(2\left(L_N + \frac{2 \times MS}{B_N}\right) + \left(L_M + \frac{2 \times DS}{B_M}\right) - 1\right) < m_b\left(2\left(L_N + \frac{MS}{B_N}\right) + \left(L_M + \frac{DS}{B_M}\right) - 1\right)$$

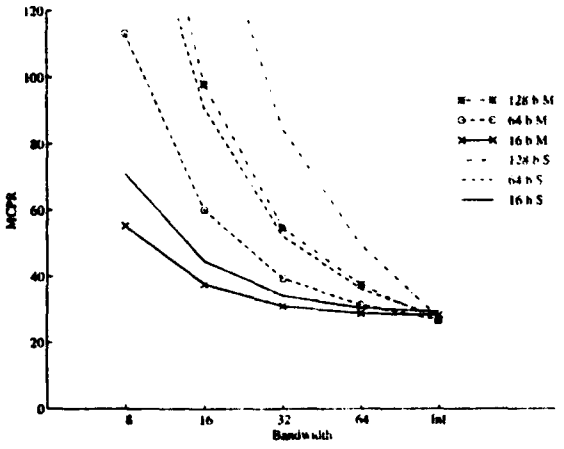

Figure 21: Simulated vs. predicted MCPR of **SOR**.

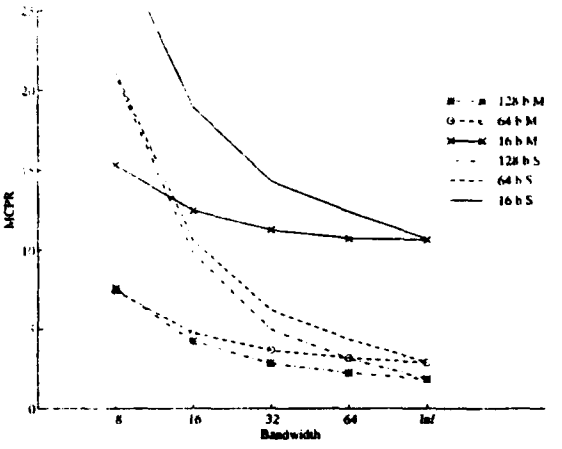

Figure 22: Simulated vs. predicted MCPR of **Gauss**.

Assuming the network and memories have comparable bandwidth (i.e, $B_N = B_M$), this simplifies to

$$m_{2b} < \frac{2MS + DS + B_N(2L_N + L_M - 1)}{4MS + 2DS + B_N(2L_N + L_M - 1)} m_b.$$

When the block size is small, the bandwidth and latency factors dominate, and this ratio is close to 1. Thus, for small block sizes, we need relatively little improvement in the miss rate to offset the higher miss penalty. As we increase the block size, both MS and DS also increase, and eventually dominate the other factors, at which point the ratio is roughly $\frac{1}{2}$. At that point, doubling the block size must cut the miss rate in half in order to lower the MCPR.

Note that this estimate of the improvement in miss rate needed to justify the next larger block size is conservative, in that it does not take into account any contention caused by using the larger cache block size. To improve the MCPR, the miss rate might have to improve by even more than is suggested by our model.

To illustrate the difficulty of justifying large blocks, consider an application with good locality of reference: **Ind Blocked LU**. Using the statistics collected via simulation under infinite bandwidth (and assuming the architecture is as specified by our simulation parameters), we find that in order to justify an increase in block size from 32 to 64 bytes, the miss rate with 64-byte blocks (3.3%) must be no more than 0.88 times the miss rate with 32-byte blocks (4.3%). Since in this case the improvement in miss rate does compensate for the increase in the miss penalty (assuming high bandwidth), an increase in block size lowers the MCPR (as seen in figure 18). A further increase in block size to 128 bytes would not be worthwhile however, since the resulting miss rate (3.1%) is not low enough (2.7% or 0.82 times the miss rate with 64-byte blocks) to justify the increase in the miss penalty.

Even programs with excellent locality may not be able to produce enough improvement in the miss rate to justify large cache blocks. For example, **Padded SOR** has excellent locality; the miss

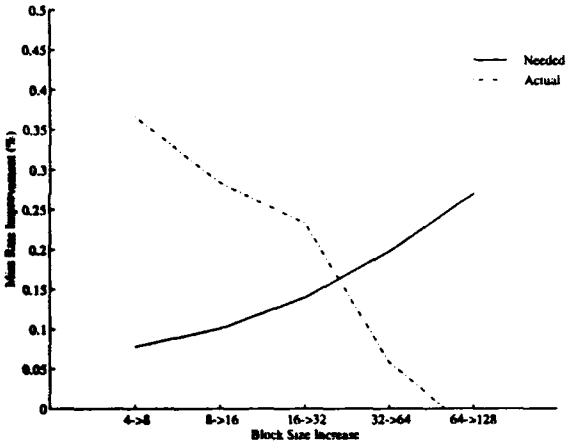

Figure 23: Actual vs. required miss rate improvement of **Barnes-Hut**

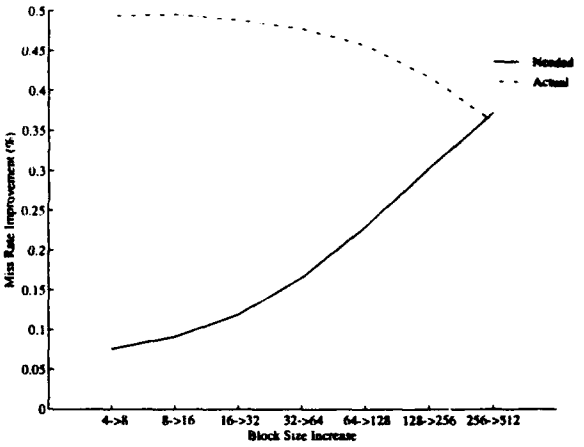

Figure 24: Actual vs. required miss rate improvement of **Padded SOR**

rate for **Padded SOR** decreases with an increase in block size up to 4K bytes. However, even though the miss rate with 512-byte blocks is very low (0.146%), and is 0.64 times the miss rate with 256-byte blocks (0.228%), it is not low enough; according to the model (and assuming high bandwidth) the ratio must be at most 0.57 to justify an increase in block size from 256 bytes to 512 bytes. Thus, the model correctly predicts that the MCPR with 256-byte blocks is lower than the MCPR with 512-byte blocks, even though the miss rate with 512-byte blocks is lower.

Figures 23-26 show the actual percentage improvement in miss rate as a function of block size compared to the percentage improvement required to offset the higher miss penalty predicted by the model under high bandwidth. These figures illustrate just how hard it is to justify large cache blocks. For **Barnes-Hut** there is a steady *decrease* in the percentage improvement in the miss rate as a function of block size, while a steady *increase* is required to justify large blocks (even under high bandwidth). The trends of ever smaller actual improvements in miss rate and ever larger required improvements eventually cross, even for programs with good spatial locality, such as **Padded SOR** and **TGauss**. The two lines cross at the point at which the improvement in the miss rate associated with larger blocks isn't enough to offset the higher miss penalty. The crossover point for **Barnes-Hut** (32 bytes), **Padded SOR** (256 bytes), and **TGauss** (128 bytes) all agree with our detailed simulations.

Mp3d2 (figure 26) is an unusual case in that the percentage improvement in the miss rate gained by moving from 32 to 64 byte blocks is higher than the percentage improvement gained by moving from 16 to 32 byte blocks. Although the actual improvement in the miss rate does not steadily decline, the required improvement does steadily rise. Thus, an increase in block size from 8 to 16 bytes is justified, but an increase from 16 to 32 bytes is not. The largest block size for which the actual miss rate improvement is at least the improvement required is 64 bytes, which is consistent with our detailed simulations.

In summary, the improvement in the miss rate required to offset an increase in miss penalty increases with the block size, but this improvement must come from an ever smaller miss rate. For practical levels of bandwidth, and for most parallel applications, the improvements in miss rate beyond 128-byte blocks are too small to offset the increase in miss penalty.

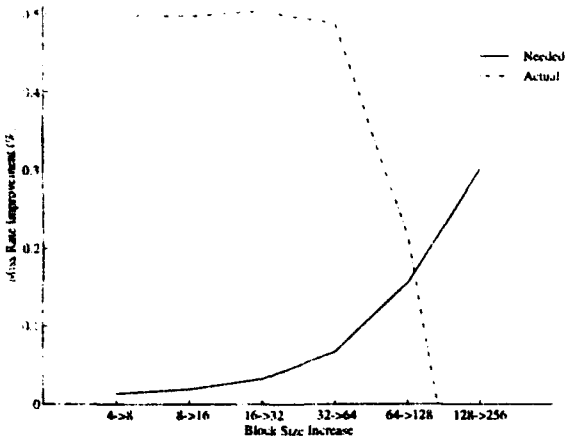

Figure 25: Actual vs. required miss rate improvement of **TGauss**

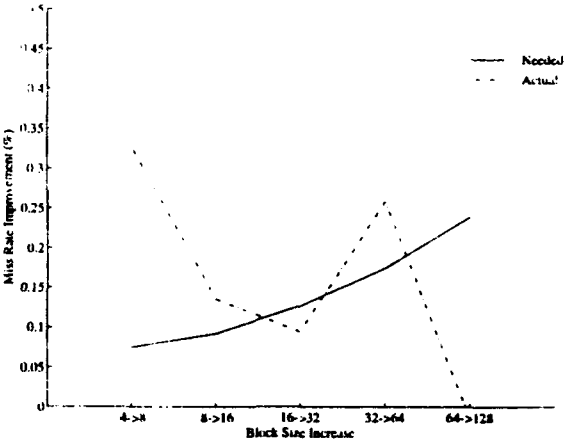

Figure 26: Actual vs. required miss rate improvement of **Mp3d2**

6.3 Implications of High Network Latency

As processor speeds continue to improve, and as we consider building larger and larger multiprocessors, we can expect remote access latency (in terms of processor cycles) to increase. Here we use our analytic model to examine the impact on MCPR of an increase in network latency.

Throughout this paper, we have assumed that network links impose a 1 cycle delay on each message, while switch nodes impose a 2-cycle delay. We will now consider four levels of network latency: *low* latency assumes delays of 0.5 and 1 cycle for the links and switch nodes respectively; *medium* latency is our original assumption for the delays; *high* latency assumes delays of 2 and 4 cycles respectively; *very high* latency assumes delays of 4 and 8 cycles respectively. Assuming infinite network and memory bandwidth, an average memory latency of 15 cycles, and an average message distance of 6 switch nodes, these latencies roughly correspond to an average remote access latency of 30, 50, 90, and 160 cycles, when moving from low to very high latency.

As a representative example of the effect of network latency on MCPR, consider **Barnes-Hut** with high or very high bandwidth. Recall that 64-byte blocks produce the minimum miss rate for **Barnes-Hut**, while 32-byte blocks produce the lowest MCPR under our earlier assumptions. As seen in figures 27 and 28, network latency has a greater impact on MCPR when small cache blocks (8 or 16 bytes) are used, since the small blocks result in a higher miss rate for **Barnes-Hut**, and each extra miss suffers from an increase in network latency. Under high bandwidth, 32-byte blocks produce the lowest MCPR regardless of network latency, although the improvement over 64-byte blocks narrows with an increase in network latency. Under very high bandwidth, the improvement over 64-byte blocks is even smaller, and disappears completely at very high network latency. In fact, for **Barnes-Hut** under very high bandwidth, an increase in network latency from high to very high increases the best block size from 32 to 64 bytes.

Our model for MCPR can be used to explain why the small improvement in miss rate that results from moving to 64-byte cache blocks is enough to offset an increase in the miss penalty under very high bandwidth and very high network latency, but not under other circumstances.

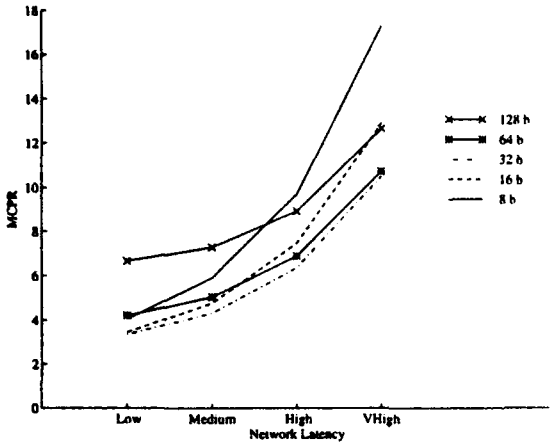

Figure 27: Predicted MCPR of **Barnes-Hut** under high bandwidth.

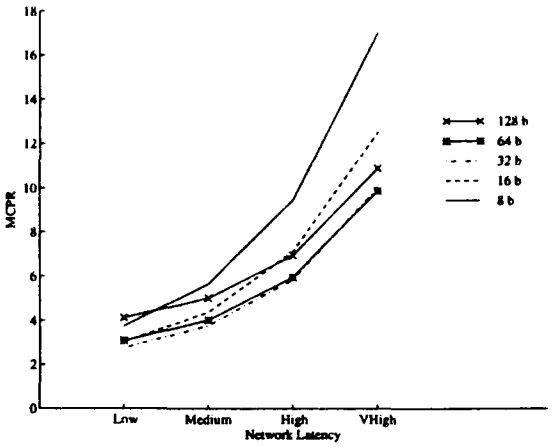

Figure 28: Predicted MCPR of **Barnes-Hut** under very high bandwidth.

Figure 29 shows the improvement in miss rate for **Barnes-Hut** needed to justify an increase in block size for our four levels of network latency (assuming high bandwidth). Whatever the network latency, larger block sizes require greater incremental improvement in the miss rate. As seen in the figure, the higher the latency, the smaller the improvement in miss rate required to justify an increase in the block size. This confirms our intuition that large blocks are not as effective with low latency as with high latency, while the reverse is true for small blocks.

Any trend in which network latency (expressed in processor cycles) continues to increase suggests a corresponding trend towards larger block sizes. The upper limit is dictated by the block size that produces the minimum miss rate, with limited bandwidth exerting downward pressure on the block size. Figures 30-32 illustrate the effects of these trends. Each figure shows the miss rate improvement actually achieved by an increase in block size for an application compared with the miss rate improvement needed to justify an increase in block size under various combinations of latency and bandwidth. Under every combination of latency and bandwidth, **Barnes-Hut** (figure 30) benefits from an increase in block size from 16 to 32 bytes. However, **Barnes-Hut** can exploit 64-byte blocks only on a machine with very high bandwidth and latency, and can never effectively exploit cache blocks larger than 64 bytes (which produce the minimum miss rate). **Mp3d** (figure 31) benefits from an increase in block size from 32 to 64 bytes under every scenario of bandwidth and latency, and can effectively exploit a further increase to 128 bytes except for the case of low latency and high bandwidth. 256-byte blocks (which produce the minimum miss rate for **Mp3d**) are only useful under very high latency and bandwidth. 256-byte cache blocks are effective for **Padded SOR** (figure 32) under all combinations of latency and bandwidth, but 512-byte blocks (which produce a lower miss rate) require very high latency to be effective. Given the trends in figure 32, even **Padded SOR** is unlikely to be able to effectively utilize cache blocks larger than 512 bytes under most realistic scenarios.

This conclusion holds even if we consider larger problem sizes. For example, if we increase the size of the problem matrix for **Padded SOR** from 384 $\times$ 384 to 512 $\times$ 512, we increase the working set size per processor from 24KB to 40KB. We also increase the block size that minimizes the miss rate from 4KB to 8KB. Nonetheless, the improvement in the miss rate beyond 512-byte blocks is

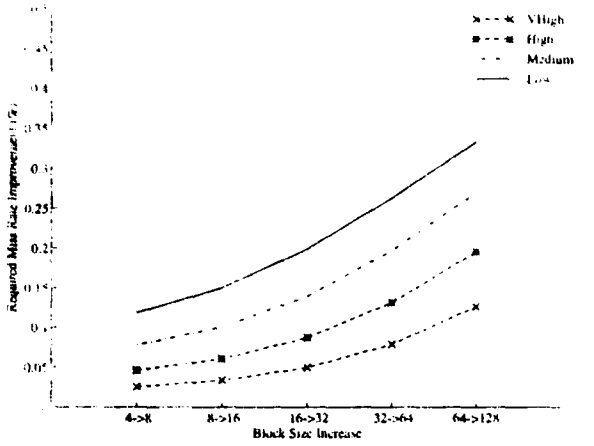

Figure 29: Predicted improvement in miss rate required to offset miss penalty for `Barnes-Hut`.

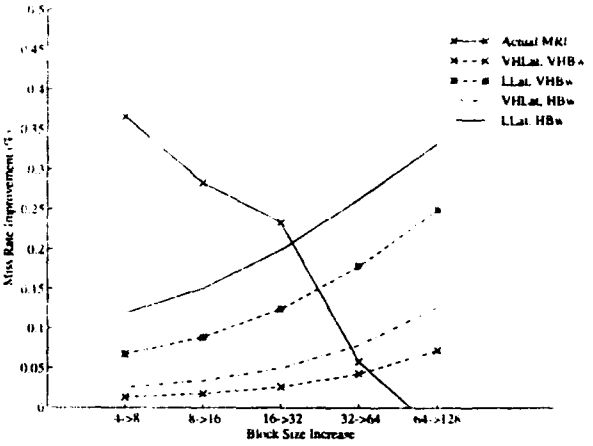

Figure 30: Actual improvement in miss rate vs. predicted improvement required for `Barnes-Hut`.

so small that larger blocks do not result in a lower MCPR except in the case of very high latency and bandwidth.

Even if the miss rate continually improves beyond 512-byte blocks, and even if the lower miss rate results in a lower MCPR, the overall effect of using larger blocks is unlikely to be significant. Under these circumstances the miss rates are so small that any improvement in miss rate has negligible impact on running time. For example, the miss rate of **Padded SOR** on an input of size 512×512 is less than 0.15% when the block size is 512 bytes. Given such a small miss rate, it would take an extremely high latency in order for a 50% improvement in the miss rate to affect application running time. Under infinite bandwidth, remote access latency would have to be as high as 250 cycles in order for blocks larger than 512 bytes to improve application performance by 10%; latency would have to be over 2000 cycles to achieve a 33% improvement in application performance.

To summarize, the block size that minimizes the miss rate is the largest block size worth considering, but the best block size depends on the bandwidth and latency of the machine. Within the range bounded by the smallest possible block size and the block size that minimizes the miss rate, bandwidth limitations argue for a decrease in block size, while high latency argues for an increase in block size. If we assume dramatic increases in both bandwidth and latency, then the best block size will approach the one that minimizes the miss rate, which is rarely larger than 256 bytes. Even if an application has sufficient locality that larger blocks reduce the miss rate, both the latency and bandwidth must be extremely high if larger blocks are to improve application performance significantly.

7 Conclusions

In this paper, we examined the relationship between cache block size and application performance as a function of remote access bandwidth and latency. Using execution-driven simulation, we

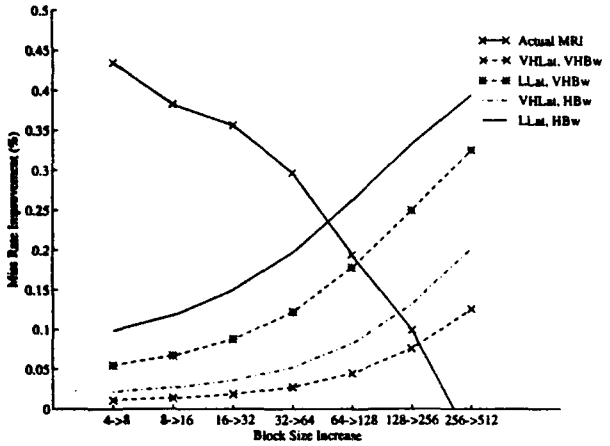

Figure 31: Actual improvement in miss rate vs. predicted improvement required for **Mp3d**.

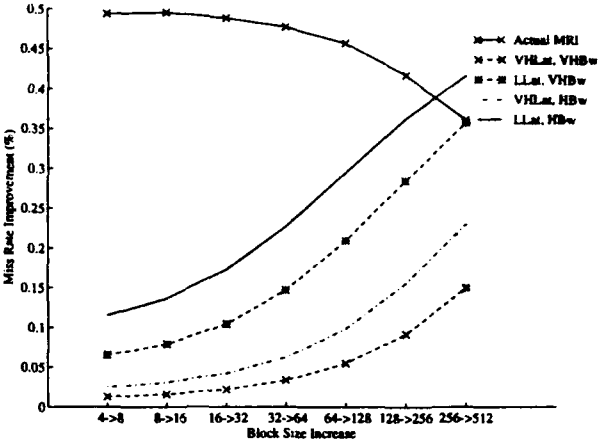

Figure 32: Actual improvement in miss rate vs. predicted improvement required for **Padded SOR**.

explored the effects of bandwidth on the choice of block size for several parallel programs using the miss rate and mean cost per reference as the primary evaluation metrics. We observed that the reference behavior of applications is such that the block size that minimizes the miss rate usually falls between 64 and 256 bytes. Nevertheless, the block size that produces the lowest mean cost per reference (assuming network latency on the order of 100 cycles and relatively high network bandwidth) usually falls between 32 and 128 bytes. Even in those cases where larger blocks produce a lower miss rate, the incremental improvement in the miss rate due to an increase in block size is often not enough to offset the increase in miss penalty associated with larger blocks.

We were surprised to see that program modifications designed to produce dramatic improvements in locality did not significantly alter our conclusions about block size. In two cases the block size that produced the minimum miss rate remained the same, while in a third case it actually got smaller. The block size that produced the lowest MCPR remained the same in one case, and grew only slightly in another case.

Using an analytical model of mean cost per reference, we showed that the *percentage improvement* in the miss rate required to offset an increase in miss penalty increases with the block size, but this improvement must come from an ever smaller miss rate. Although less improvement is required for higher latency, the actual improvement in the miss rate gained by doubling the block size steadily declines, while the improvement required to offset the miss penalty steadily rises. Our analysis suggests that for most practical levels of bandwidth and latency, and for most parallel applications, the improvements in miss rate beyond 128-byte blocks are too small to offset the increase in miss penalty. We conclude that larger blocks are justified only under extreme circumstances, such as when the remote access bandwidth and latency are both very high, and applications exhibit nearly perfect locality and very limited sharing.

Acknowledgements

We would like to thank Jack Veenstra for helping develop and improve our applications and simulation infrastructure, and Mark Crovella and Leonidas Kontothanassis for their many comments on this work. We would also like to thank Michael Marchetti for some early discussions and results.

References

[Agarwal, 1991] A. Agarwal, "Limits on Interconnection Network Performance," *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, Oct 1991.

[Agarwal and Gupta, 1988] A. Agarwal and A. Gupta, "Memory Reference Characteristics of Multiprocessor Applications under Mach," *Performance Evaluation Review*, 16(1):215–225, May 1988, originally published at SIGMETRICS '88.

[Cheriton *et al.*, 1991] David R. Cheriton, Hendrik A. Goosen, and Philip Machanick, "Restructuring a Parallel Simulation to Improve Cache Behavior in a Shared-Memory Multiprocessor: A First Experience," In *Proceedings of the International Symposium on Shared-Memory Multiprocessing*, pages 109–118, Tokyo, Japan, April 1991.

[Dackland *et al.*, 1992] K. Dackland, E. Elmroth, B. Kagstrom, and C. Van Loan, "Parallel Block Matrix Factorizations on the Shared-Memory Multiprocessor IBM 3090 VF/600J," *The International Journal of Supercomputer Applications*, 6(1):69–97, Spring 1992.

[Dubnicki, 1993] C. Dubnicki, *The Effects of Block Size on the Performance of Coherent Caches in Shared-Memory Multiprocessors*, PhD thesis, Department of Computer Science, University of Rochester, July 1993.

[Dubois *et al.*, 1993] M. Dubois, J. Skeppstedt, L. Ricciulli, K. Ramamurthy, and P. Stenstrom, "The Detection and Elimination of Useless Misses in Multiprocessors," In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 88–97, San Diego, CA, May 1993.

[Eggers and Jeremiassen, 1991] S. J. Eggers and T. E. Jeremiassen, "Eliminating False Sharing," In *Proceedings 1991 International Conference on Parallel Processing*, pages 377–381, St. Charles, IL, August 1991.

[Eggers and Katz, 1989] S. J. Eggers and R. H. Katz, "The Effect of Sharing on the Cache and Bus Performance of Parallel Programs," In *Proceedings of the 3rd International Conference on Architectual Support for Programming Languages and Operating Systems*, pages 257–270, Boston, MA, April 1989.

[Gupta and Weber, 1992] A. Gupta and W.-D. Weber, "Cache Invalidation Patterns in Shared-Memory Multiprocessors," *IEEE Transaction on Computers*, 41(7):794–810, July 1992.

[LeBlanc, 1988] T.J. LeBlanc, "Problem Decomposition and Communication Tradeoffs in a Shared-Memory Multiprocessor," In Martin Schultz, editor, *Numerical Algorithms for Modern Parallel Computer Architectures, IMA Volumes in Mathematics and its Applications Volume 13*, pages 145–163. Springer-Verlag, 1988.

[Lee *et al.*, 1987] R. L. Lee, P.-C. Yew, and D. H. Lawrie, "Multiprocessor Cache Design Considerations," In *Proceedings of the 14th Annual Symposium on Computer Architecture*, pages 253–262, Pittsburgh, PA, June 1987.

[Lenoski *et al.*, 1990] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor," In *Proceedings of*

the 17th International Symposium on Computer Architecture, pages 148–159, Seattle, WA, May 1990.

[Przybylski, 1990] S. Przybylski, "The Performance Impact of Block Sizes and Fetch Strategies," In *Proceedings of the 17th Annual Symposium on Computer Architecture*, pages 160–169, Seattle, WA, 1990.

[Singh *et al.*, 1992] J.P. Singh, W-D. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory," *Computer Architecture News*, 20(1):5–44, March 1992.

[Smith, 1987] A. J. Smith, "Line (Block) Size Choice for CPU Cache Memories," *IEEE Transactions on Computers*, C-36(9):1063–1075, September 1987.

[Veenstra, 1993] J. E. Veenstra, "Mint Tutorial and User Manual," Technical Report 452, Department of Computer Science, University of Rochester, July 1993.